



Java PathFinder

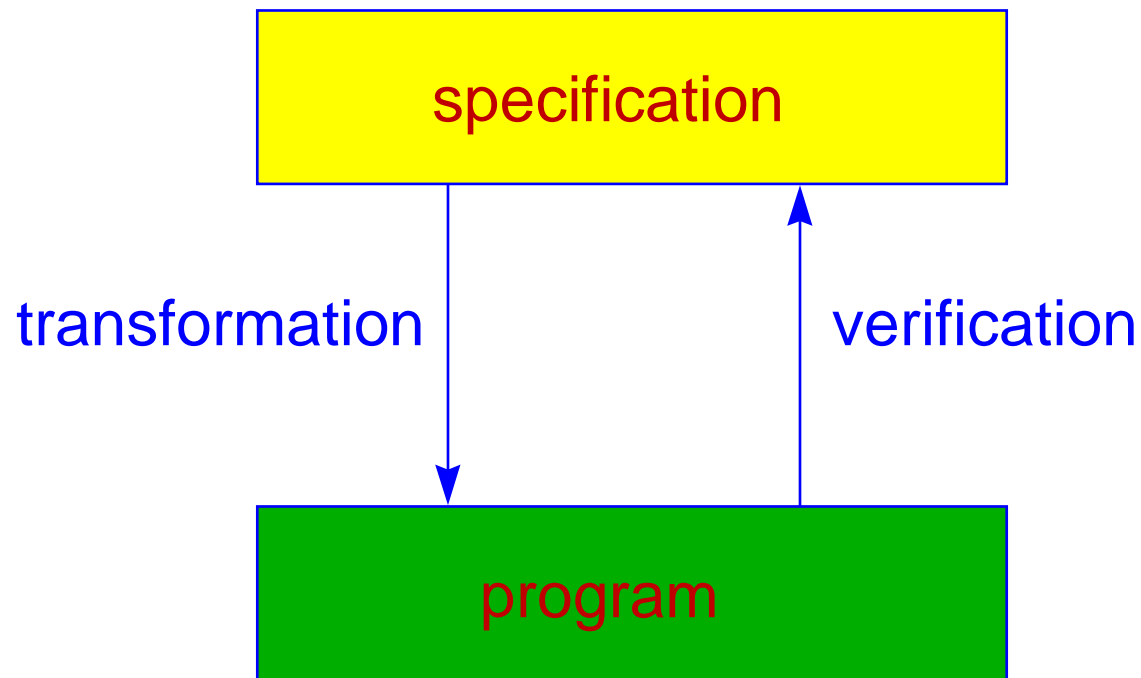
K. Havelund

Recom Technologies

NASA Ames Research Center

California, USA

Automated Software Engineering



<http://ic-www.arc.nasa.gov/ic/projects/amphion>

Current ASE Projects

Transformation

- *Solar geometry*
- *Data Analysis*
- *Grid generation*
- *Reuse*

Verification

- *Program verification (JPF, DEOS)*
- *Design verification (UML, Control Shell)*
- *Autonomy Software (Livingstone, Plans)*

The Correctness Problem

- *Trust me*
- *Because I designed it*
- *It is obviously correct*
- *It has not failed in the last N years*
- *It has been tested*
- *All the demo's work perfectly*



Traditional Methods

- *Peer review*
- *Debugging*
- *Random simulation*
- *Exhaustive testing*
- *Proof?*

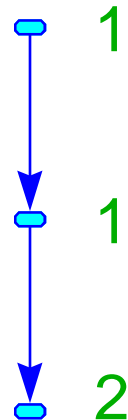
Two Sequential Programs

A:

if state == 1 then
state++

state == 1

state++

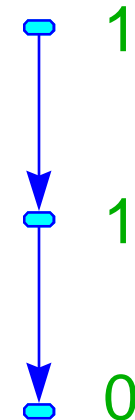


B:

if state == 1 then
state--

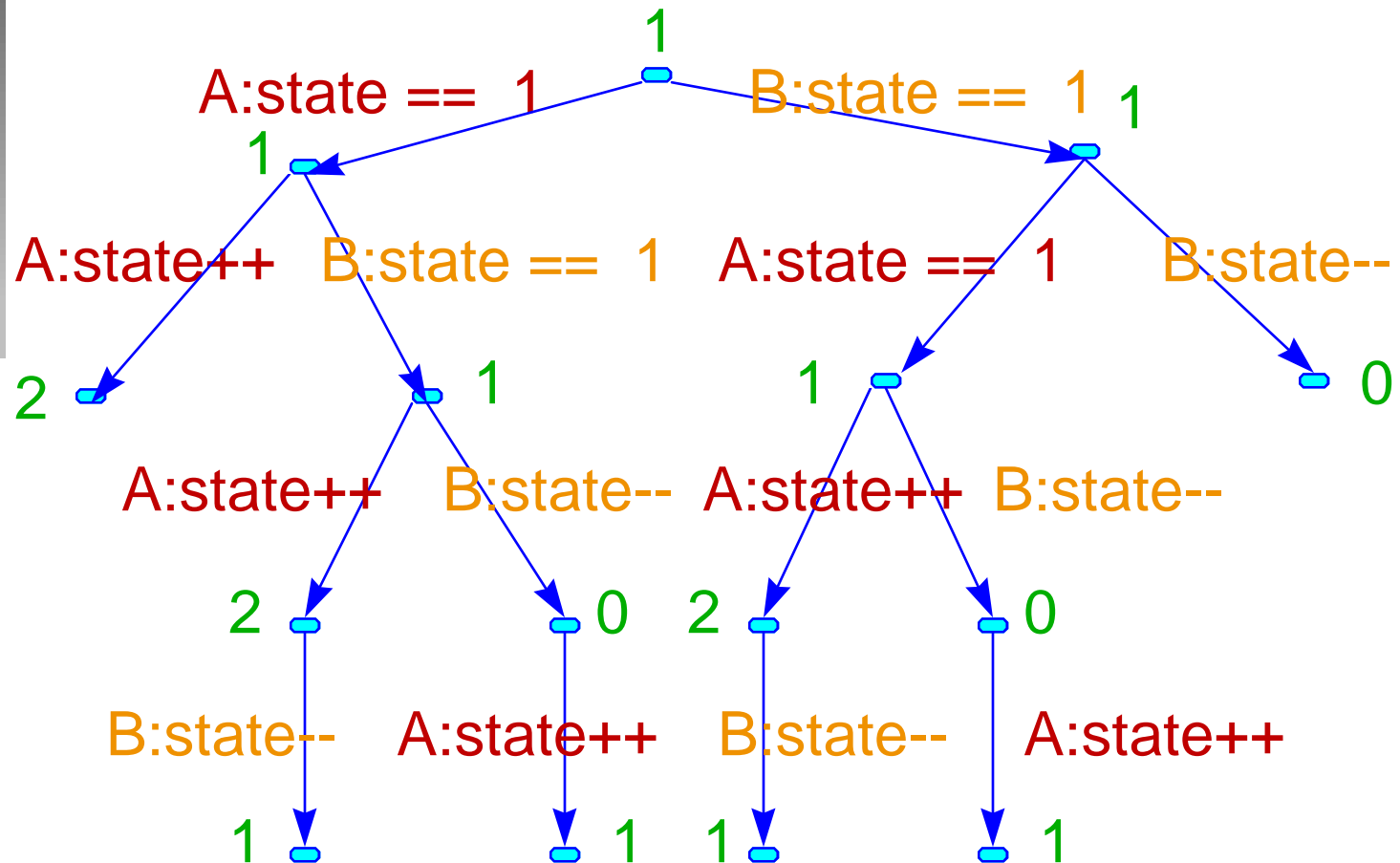
state == 1

state--



Initial state == 1

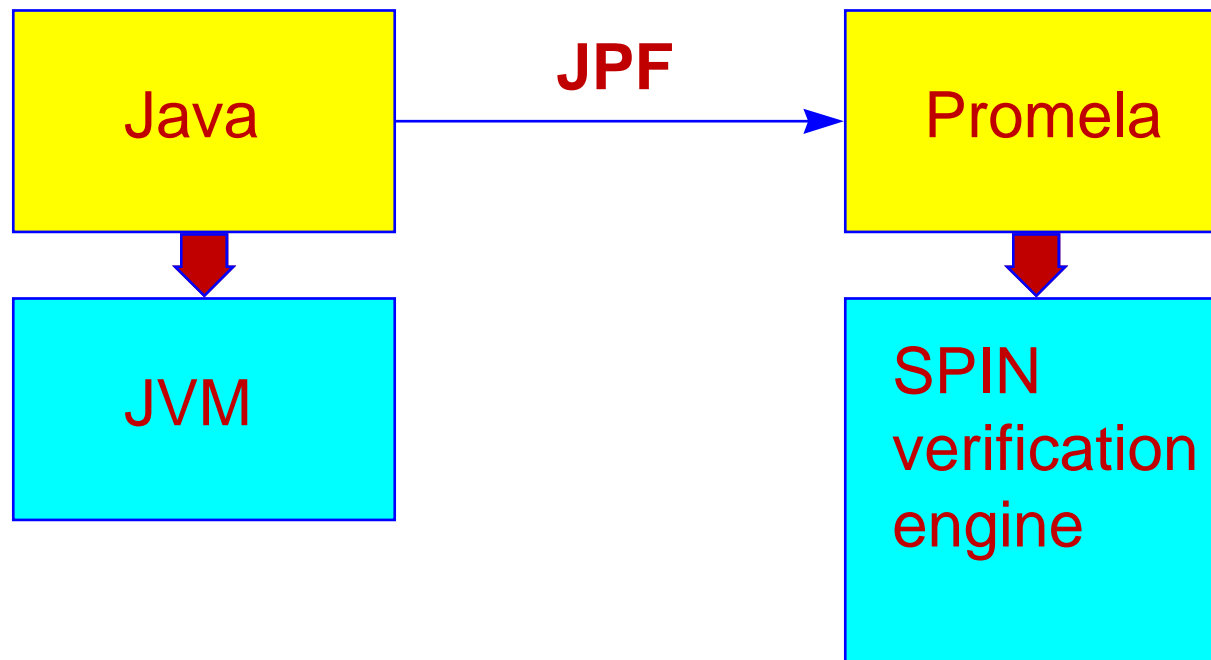
The Concurrency Syndrom



The Concurrency Test problem

- *Not enough to control test **inputs***
- *Also requires control over **scheduler***
- *Even worse : scheduler may differ between platforms*

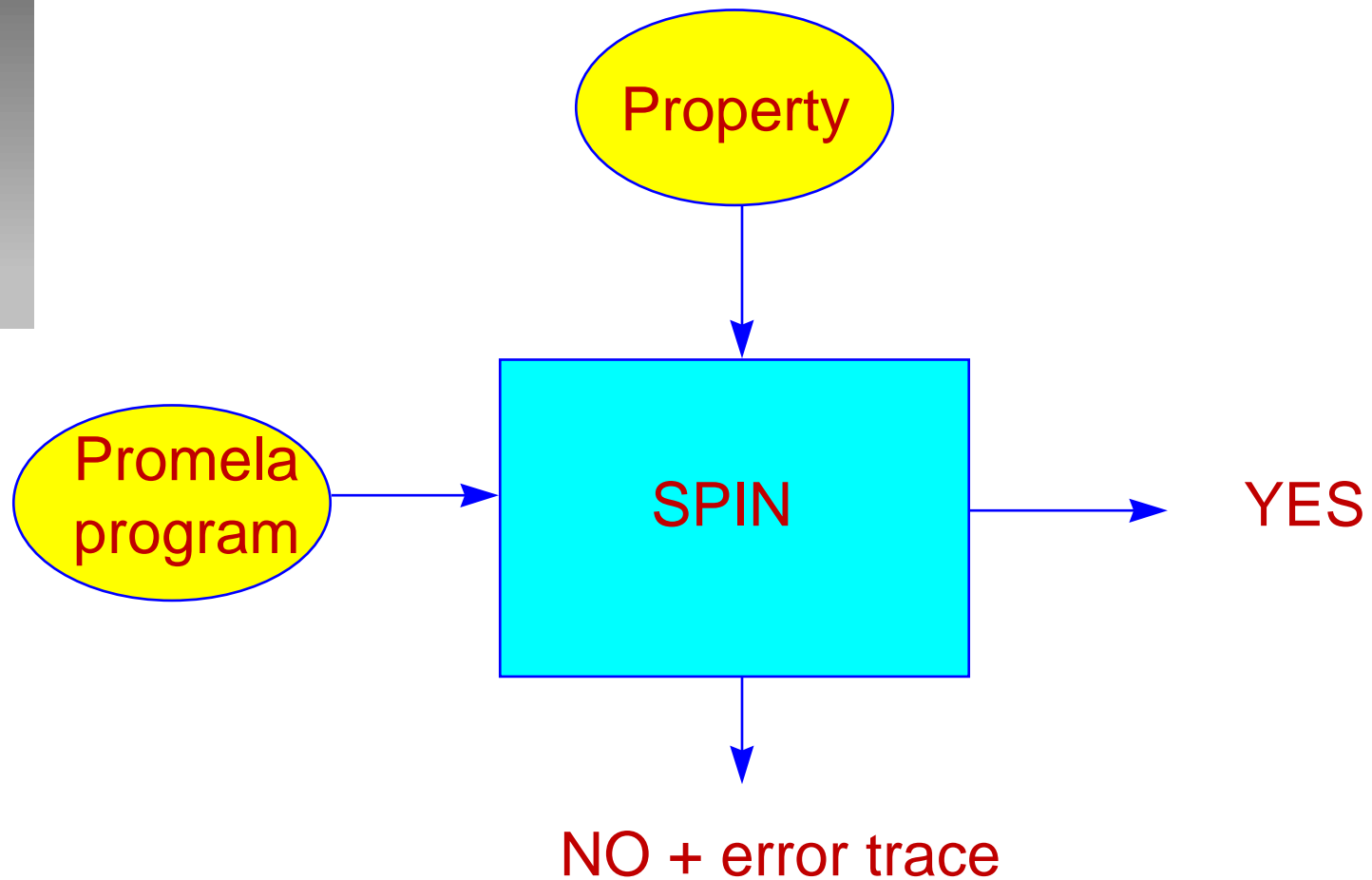
JPF : Java -> SPIN (Promela)



Rest of Talk

- *SPIN*
- *JPF*
- *Remote Agent Experience (using SPIN)*
- *“Release Lock” Problem in JPF*

The SPIN Tool





Promela

Three types of objects

- *Processes*
- *Variables*
- *Message Channels*

Example Promela Program

```
byte state = 1;
```

```
proctype A()  
{state == 1 -> state++; assert(state==2)}
```

```
proctype B()  
{state == 1 -> state--; assert(state == 0)}
```

```
init{run A();run B()}
```

Error Trace - Assertion Violated

byte state = 1
run A()
run B()

state == 1

state == 1

state-- (0)

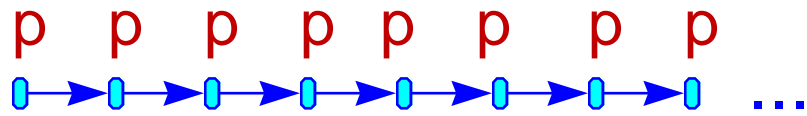
assert(state == 0)

state++ (1)

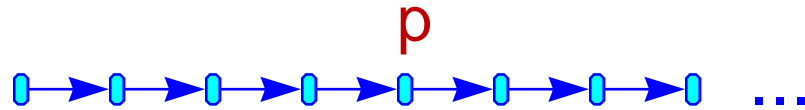
assert(state == 2)

Linear Temporal Logic (LTL)

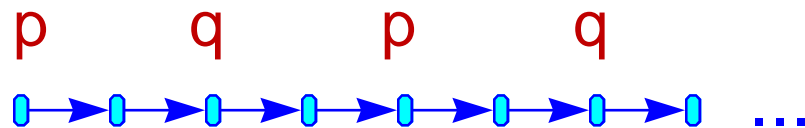
$[]p$



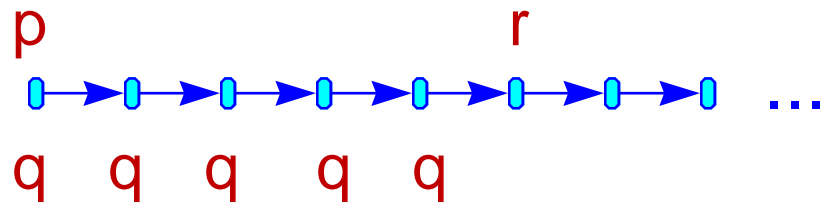
$<>p$



$[](p \rightarrow <> q)$



$[](p \rightarrow q \cup r)$



Deadlocks will also be detected

Another Example

```
int incrit;  
bool flag[2];  
bool turn;
```

```
#define CRITICAL incrit++;incrit--  
#define other (1 - this)
```

```
proctype user(byte this){  
    flag[this] = true;  
    turn = this;  
    flag[other] == false || turn == other;  
    CRITICAL;  
    flag[this] = false  
}
```

```
init{run user(0); run user(1)}
```

LTL Property:

```
[](incrit == 0 ||  
    incrit == 1)
```


SPIN's Verification Engine

- *Theoretical foundation : Buchi automata*
- *Efficient state storage (states are remembered to avoid redundancy)*
- *Partial order reduction to cut down number of examined execution traces*
- *Bitstate hashing as an **approximation** when everything else fails*

The Java Pathfinder

- *Write your program/design in Java*
- *Write specifications using special methods from the **Verify** class.*
- *Press the button*

The Verify Class

```
class Verify{  
    public static void assert(boolean b){  
    }  
  
    public void methodA(){  
        if (shared.state == 1){  
            shared.state++;  
            Verify.assert(shared.state == 2);  
        };  
    }  
}
```

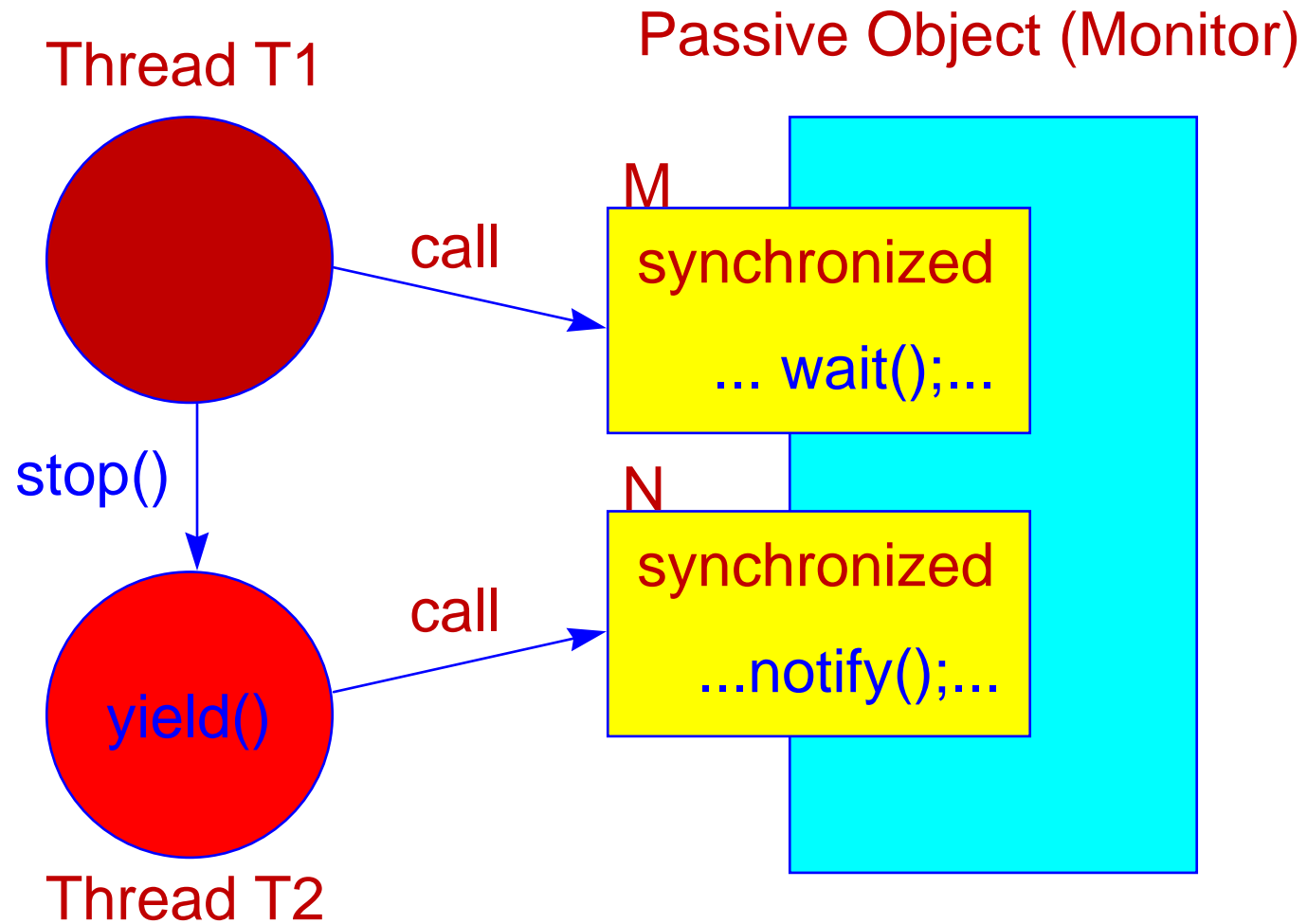


Threads in Java

```
class P extends Thread{  
    ...  
    public void run(){...}  
}
```

```
P p = new P();  
p.start();
```

Communication via Monitors



File Transfer

```
class Transfer extends Thread{
    File from,to;

    public Transfer(File from,File to){
        this.from = from; this.to = to; this.start();
    }

    public void run(){
        synchronized(from){
            synchronized(to){
                from.transfer(to)
            }
        }
    }
}
```

File Transfer (Cont.)

```
class Main{  
    public static void main(String[] args){  
        File f1 = new File(...);  
        File f2 = new File(...);  
        Transfer t1 = new Transfer(f1,f2);  
        Transfer t2 = new Transfer(f2,f1);  
    }  
}
```

Error Trace : Deadlock

Transfer(f1,f2)

locks f1

waits on f2

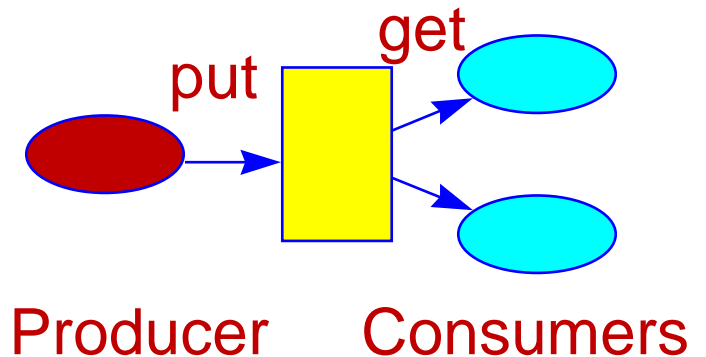
Transfer(f2,f1)

locks f2

waits on f1

(Put Get Put Get ...) Pattern

```
class CubbyHole{  
    private boolean available = false;  
    public int balance = 0;  
  
    public synchronized void get(){  
        if (available == false){  
            try{wait();} catch (Exception e){}  
        };  
        available = false;  
        balance--;  
        notify();  
    }  
}
```





*(Put Get) **

```
public synchronized void put(){  
    if (available == true){  
        try{wait();} catch (Exception e){}  
    };  
    available = true;  
    balance++;  
    notify();  
}  
}
```

*(Put Get)**

```
class Main{  
    public static void main(String[] args){  
        CubbyHole c      = new CubbyHole();  
        Producer prod    = new Producer(c);  
        Consumer cons1 = new Consumer(c);  
        Consumer cons2 = new Consumer(c);  
        Verify.assert(c.balance == 0 ||  
                       c.balance == 1);  
    }  
}
```

Error Trace : Assertion Violated

Producer

put() - notify()

1

Consumer 1

get() - wait()

0

notify()

Consumer 2

get() - wait()

-1

Repair

```
class CubbyHole{  
    private boolean available = false;  
    public int balance = 0;  
  
    public synchronized void get(){  
        while (available == false){  
            try{wait();} catch (Exception e){}  
        };  
        available = false;  
        balance--;  
        notify();  
    }  
}
```



Mutual Exclusion

```
class Sem{  
    public boolean[] flag = new boolean[2];  
    public int turn;  
    public int incrit = 0;  
}
```

Mutual Exclusion (Cont.)

```
class Process extends Thread{  
    private Sem sem;  
    private int pid;  
  
    public Process(Sem sem,int pid){  
        this.sem = sem;  
        this.pid = pid;  
        this.start();  
    }  
}
```

... continued ...

Mutual Exclusion (Cont.)

```
public void run(){
    while (true){
        sem.flag[pid] = true;
        sem.turn = pid;
        while (!(sem.flag[1-pid] == false || sem.turn == 1-pid)){};
        sem.incrit++;
        Verify.assert(sem.incrit == 1);
        sem.incrit--;
        sem.flag[pid] = false;
    };
}
```


Mutual Exclusion (Cont.)

```
class Main{  
    public static void main(String[] args){  
        Sem sem = new Sem();  
        Process p1 = new Process(sem,0);  
        Process p2 = new Process(sem,1);  
    }  
}
```



No Errors

- *Assertion is not violated*
- *There are no deadlocks*

More “Natural” Solution ?

```
public void run(){
    while (true){
        sem.flag[pid] = true;
        sem.turn = pid;
        while (!(sem.flag[1-pid] == false && sem.turn == pid)){
            sem.incrit++;
            Verify.assert(sem.incrit == 1);
            sem.incrit--;
            sem.flag[pid] = false;
        };
    }
}
```

before:

|| sem.turn == 1-pid

No Errors, Correct??

- *Assertion is not violated*
- *There are no deadlocks*
- *Let's try : `[](incrit==0 -> <>incrit==1)`*

“Natural” Solution (Cont.)

```
class Main{  
    public static void main(String[] args){  
        Sem sem = new Sem();  
        Process p1 = new Process(sem,0);  
        Process p2 = new Process(sem,1);  
        Verify.response(sem.incrit == 0,  
                        sem.incrit == 1);  
    }  
}
```

Error Trace : Livelock

Process 0

flag[0] = true
turn = 0

busy wait until:
flag[1] == false
& turn == 0

Process 1

flag[1] = true
turn = 1

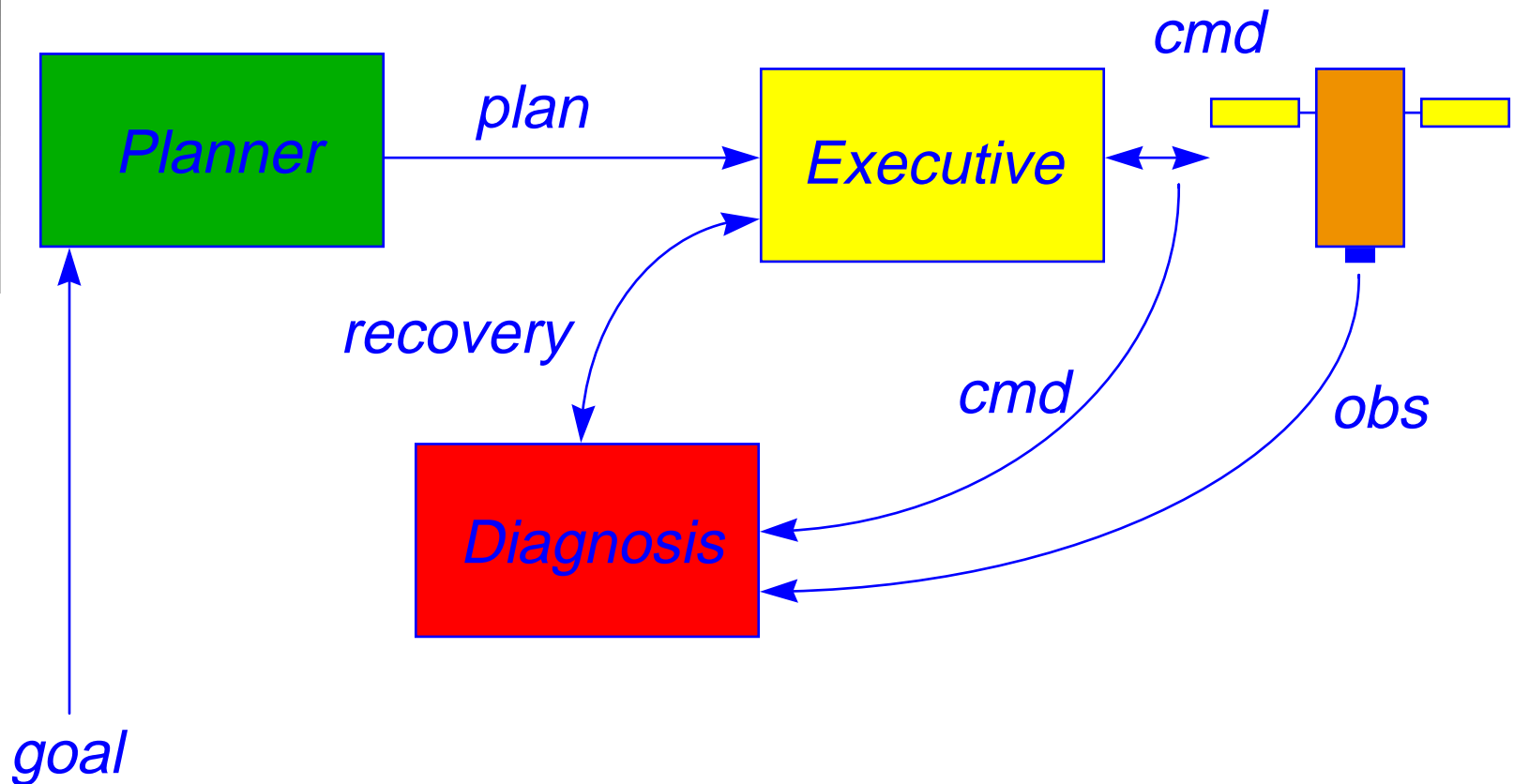
busy wait until:
flag[0] == false
& turn == 1



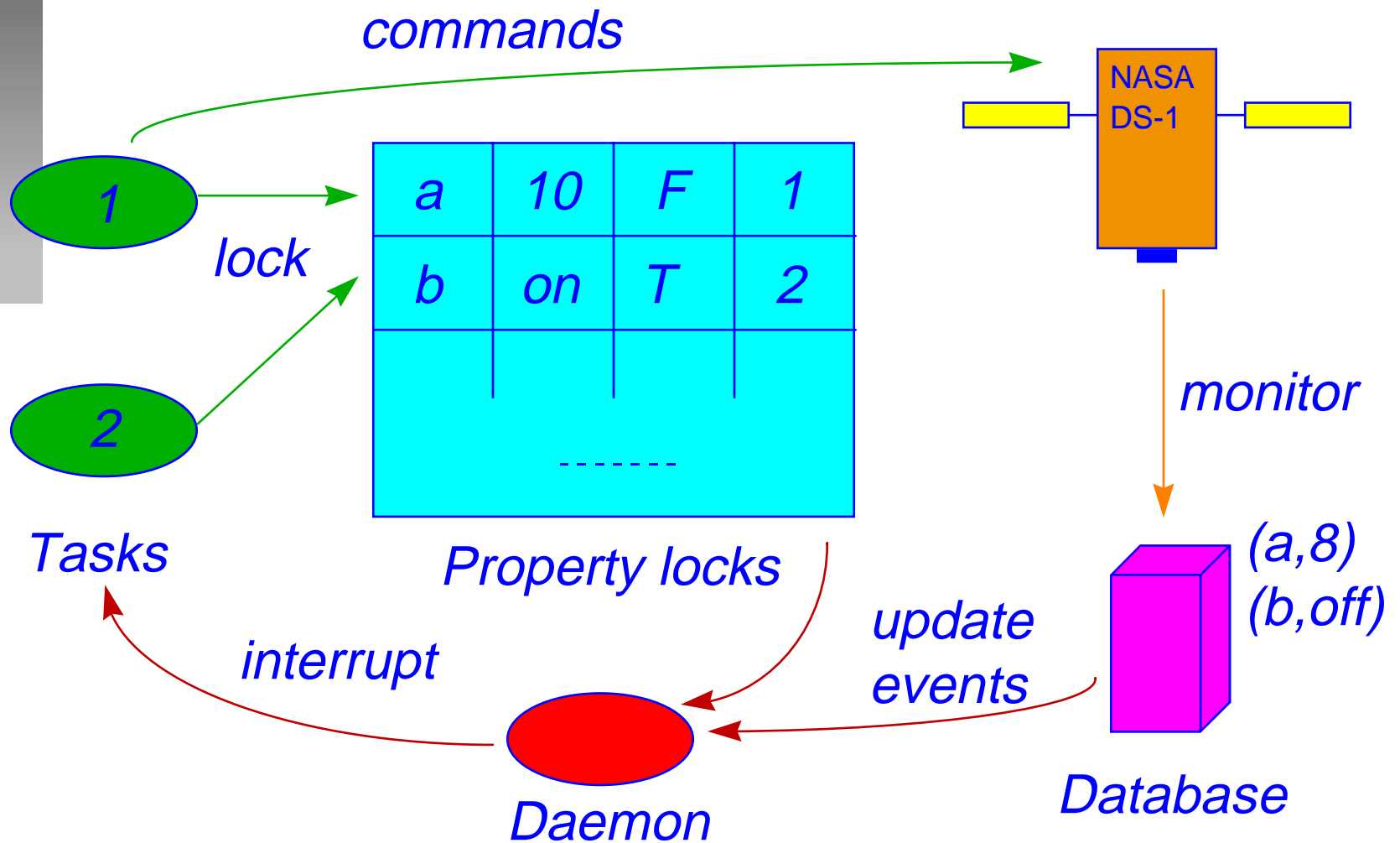
Formal Analysis of Deep Space 1

K. Havelund, M. Lowry, J. Penix
NASA Ames Research Center
California, USA

The Remote Agent



Informal Description of Exec.



Task Execution

```
inline execute_task(this,p){  
    bool err = 0;  
    {snarf_property_lock(this,p,err);  
      achieve_lock_property(this,p,err);  
      closure()  
    } unless {err || active_tasks[this].state == ABORTED};  
  
    active_tasks[this].state = TERMINATED;  
  
    {release_lock(this,p)} unless  
      {active_tasks[this].state == ABORTED}  
}
```



Properties Stated by Engineers

Release Property:

A task releases all its locks before it terminates.

Abort Property:

If an inconsistency occurs between the database and an entry in the lock table, then all tasks that rely on the lock will be terminated, either by themselves or by the daemon.

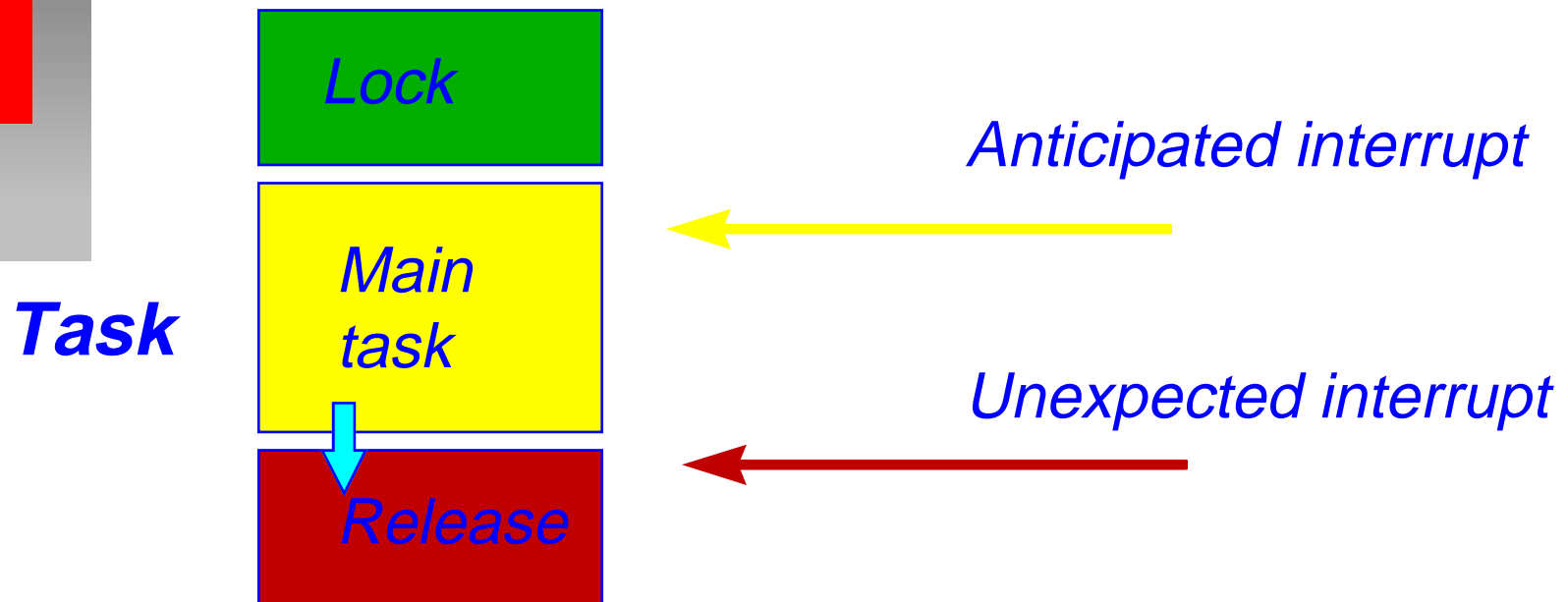
The Release Property in SPIN

```
#define not_subscriber(this,pn)
    !locks[pn].sub??[eval(this)]

proctype Achieving_Task(TaskId this){
    ...
    execute_task(this,p);
    assert(not_subscriber(this,p.name))
}
```

in Task

Error 1 : Lock Releasing




↓ = *Unwind-protect in LISP*

in Task

Error 1 : the Promela Code

```
inline execute_task(this,p){  
  bool err = 0;  
  {snarf_property_lock(this,p,err);  
   achieve_lock_property(this,p,err);  
   closure()  
  } unless {err || active_tasks[this].state == ABORTED};  
  
  active_tasks[this].state = TERMINATED;  
  
  {release_lock(this,p)} unless  
    {active_tasks[this].state == ABORTED}  
}
```



The Abort Property

```
#define task1_property_broken  
  (locks[0].value == 1 &  
   locks[0].achieved &  
   db[0] == 0)
```

```
#define task1_terminated  
  (active_tasks[1].state == TERMINATED ||  
   active_tasks[1].state == ABORTED)
```

[!](task1_property_broken -> <>task1_terminated)

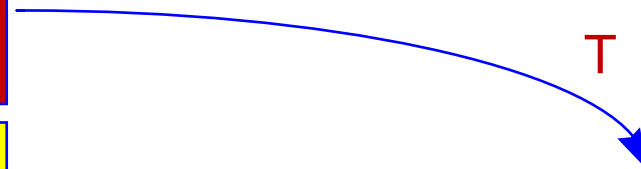
in Task

Error 2 : Achieve Procedure

**Main
Task**



Property broken



a	10	F	1

in Task

Error 2 Repair : Critical Section

**Main
Task**



Introduce critical section

Error 3 : Check Locks Procedure

Check Locks

*Check locks :
interrupt*

- no change discovered
- no interrupts

Unexpected change



*Check locks :
repair remains*

- change discovered
- automatic recovery
- **but NO interrupts**

Error 3 “Repair” : New Property

We did initially not regard this as an error!

IT WAS THOUGH!

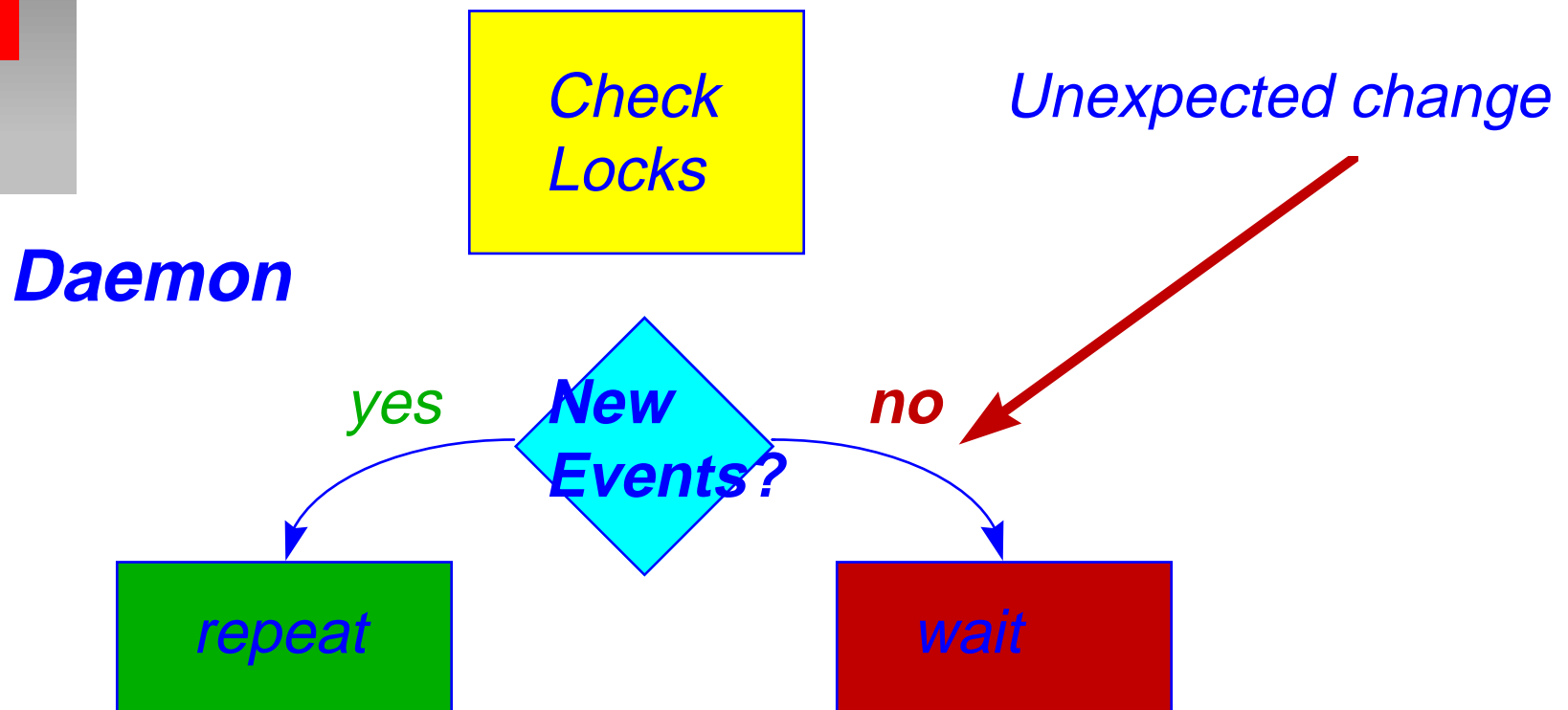
Therefore we modified correctness property:

```
#define task1_property_repaired  
locks[0].value == db[0]
```

```
[](task1_property_broken ->  
  <> (task1_terminated ||  
      task1_property_repaired))
```

in Daemon

Error 4 : Waiting Procedure

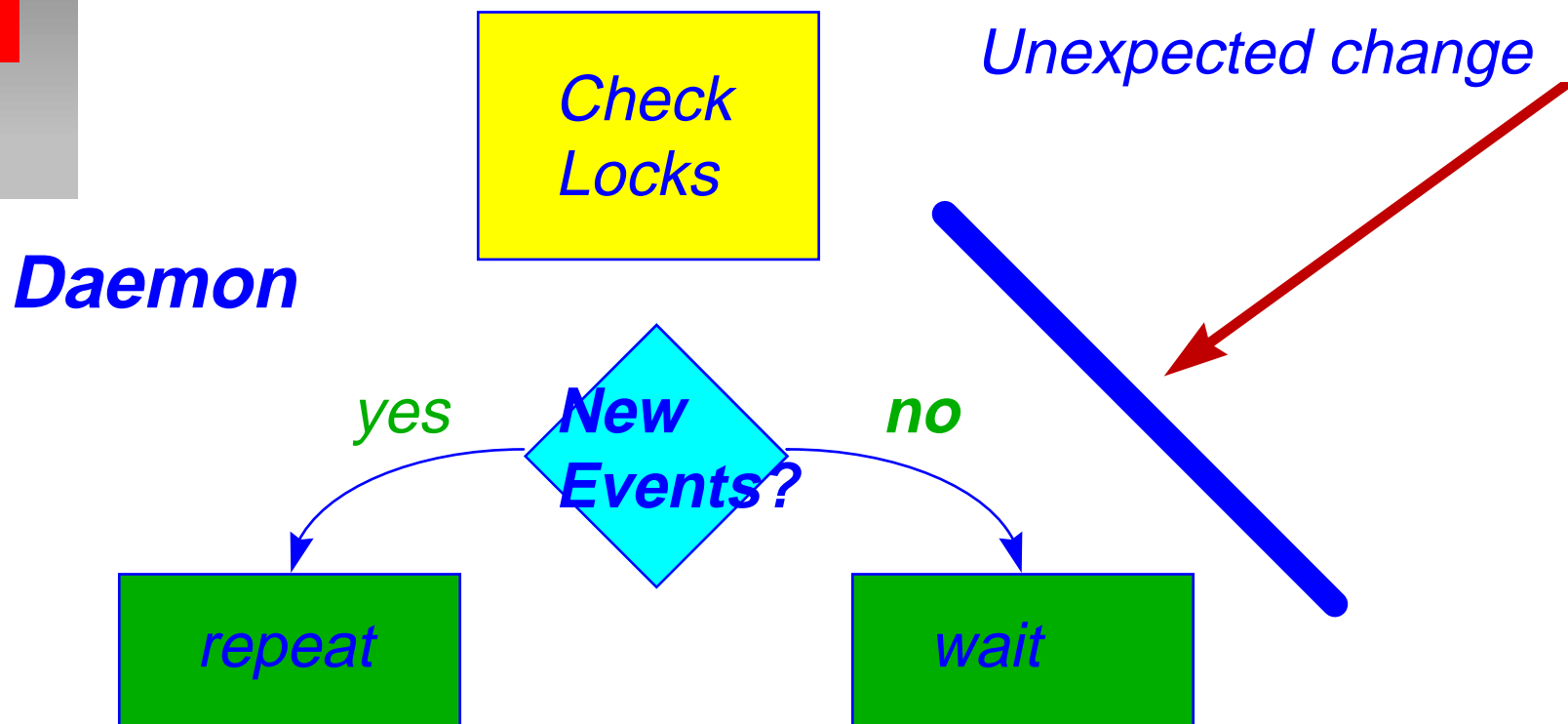


Error 4 : the Promela Code

```
proctype Daemon(TaskId this){  
  bit lock_violation;  
  do  
    :: check_locks(lock_violation);  
    "if lock_violation do automatic recovery";  
    if  
      :: "event counters changed" -> "repeat"  
      :: else -> wait_for_events(this)  
    fi  
  od  
}
```

in Daemon

Error 4 Repair: Critical Section

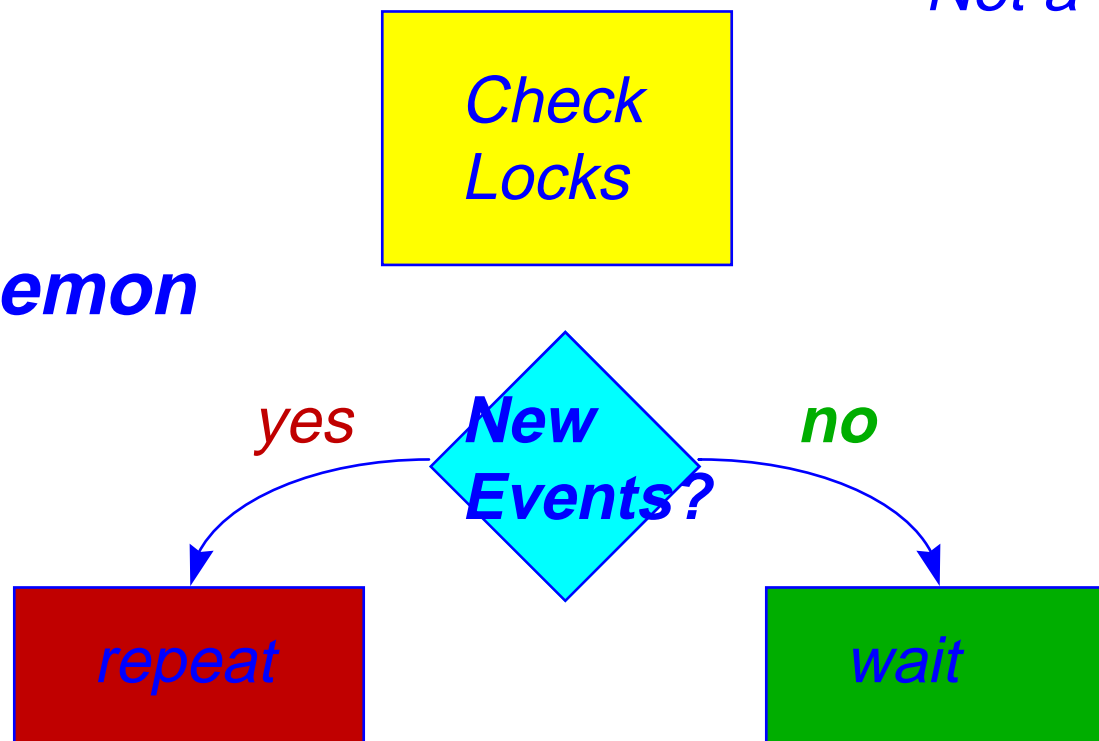


in Daemon

Error 5 : Extra Lock Check

Not a serious error

Daemon



At least one extra check

Programmer's Comment to Err.1

“ ...

It arises only because you are in a multi-threaded environment, and only in very obscure circumstances that are unlikely to arise during testing.

Congratulations! You have just converted me into a believer in formal methods.”



Programmer's Comment to Err.2

“Ah, good point. You are correct, this is a bug. I’m impressed! This makes two bugs you guys have discovered through formal methods that we almost certainly would never have caught any other way.”

etc.



Did our Work have any Impact?

“You’ve found a number of bugs that I am fairly confident would not have been found otherwise. One of the bugs revealed a major design flaw (which has not been resolved yet). So I’d say you have had a substantial impact. If nothing else you have helped us improve the quality of our product well beyond what we otherwise would have produced.”

Programmer



What are your Conclusions?

“I used to be very skeptical of the utility of formal methods. This is at least partly due to the fact that I had a misconception about the way in which formal methods would be used. I thought that formal-methods advocates wanted to prove correctness of software systems. I believed (and still believe) that this is impossible. However, what you have been doing is finding places where software violates design assumptions, which is not the same thing as proving correctness. To me you have demonstrated the utility of this approach beyond any question.”

Programmer

Conclusion

Abstraction

Promela

Translation

LISP

The Release Property in JPF

```
class LockTable{
    Task[] table = new Task[3];

    public synchronized void lock(int prop, Task task)
        throws LockException{
        if (table[prop] != null){
            throw(new LockException());
        };
        table[prop] = task;}

    public synchronized void release(int prop){
        table[prop] = null;}
}
```

Release Property (Tasks)

```
class Task extends Thread{  
    LockTable t;  
    int        p;  
    Activity   a;  
  
    public Task(LockTable t,int p,Activity a){  
        this.t = t; this.p = p; this.a = a;  
        this.start();  
    }  
}
```

... continued ...

Release Property (Tasks cont.)

```
public void run(){  
    try{  
        t.lock(p,this);  
        a.activity();  
    }  
    catch (LockException e){}  
    finally {release(p);}  
}  
}
```

Release Property (Daemon)

```
class Daemon extends Thread{  
    Task task;  
  
    public Daemon(Task task){  
        this.task = task;  
        this.start();  
    }  
  
    public void run(){  
        task.stop();  
    }  
}
```


Release Property (Main)

```
class Main{
    public static void main(String[] args){
        LockTable table    = new LockTable();
        Activity    activity = new Activity();
        Task        task     = new Task(table,1,activity);
        Daemon    daemon = new Daemon(task);
        try {task.join();} catch (InterruptedException e){};
        Verify.assert(table.table[1] == null);
    }
}
```

Error Trace : Assertion Violated

- *Task is started*
- *Daemon is started*
- *Task executes activity*
- *Task enters **finally** construct, **ready** to release property*
- *Daemon stops the task*
- *Task stops and leaves **finally** construct, **no property is released***

“Hunting” error without SPIN

```
public void run(){  
    try{  
        t.lock(p,this);  
        a.activity();  
    }  
    catch (LockException e){}  
    finally {yield();release(p);};  
}  
}
```

The yield() statement
will provoke the error
to occur.



But one has to know where to hunt!

Chinese Check Program

- *Analyzed by Jens Skakkebaek (Stanford University)*
- *<http://www.cchess.net>*
- *1400 LOC, 16 classes*
- *Deadlock confirmed*
- *Shorter error trace found*

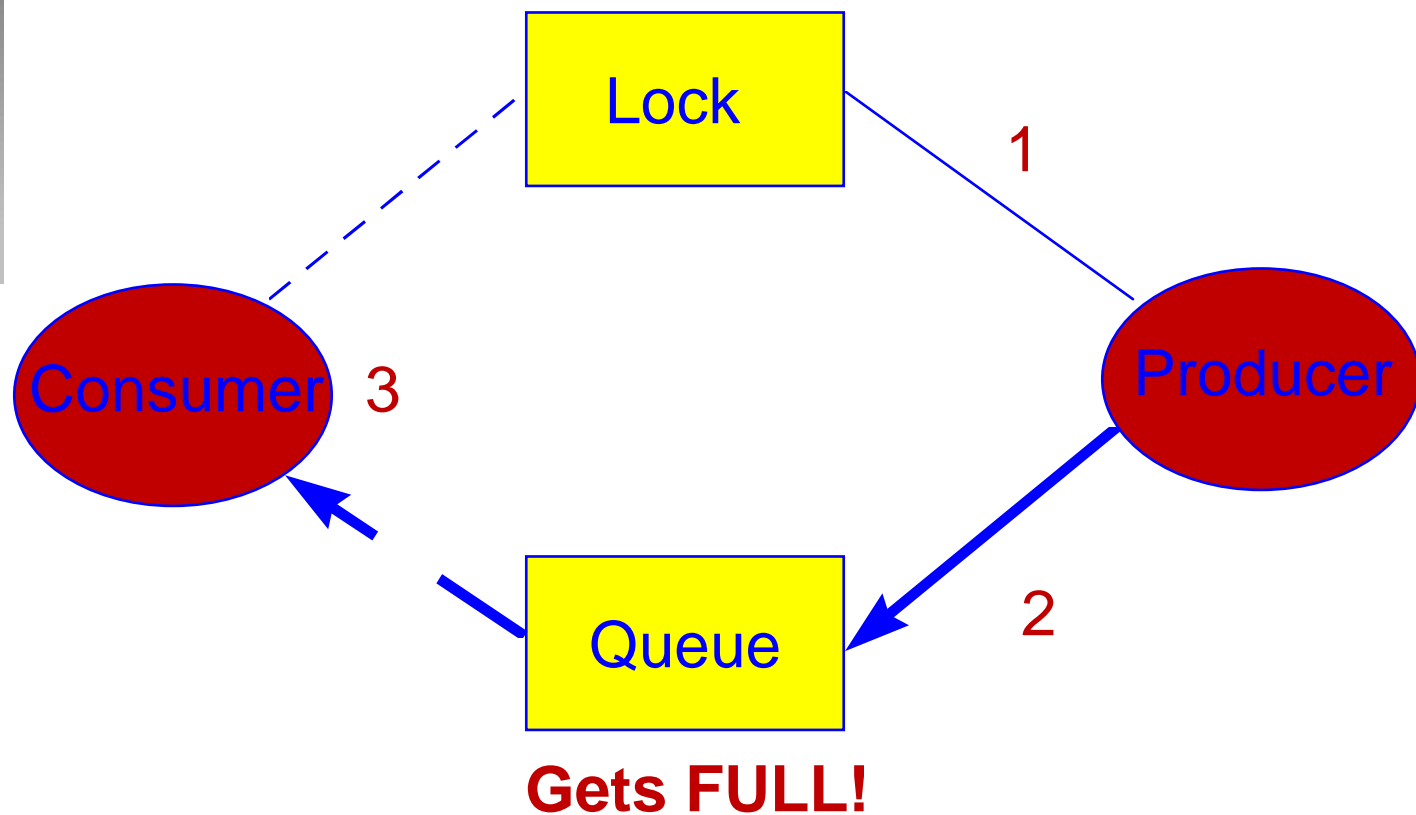


Programmer's Email

“I did my best to verify that no possible deadlock can happen, yet there is no way for me to check, and I’m likely to have made a mistake somewhere.

The program sometimes finds itself in deadlock position, probably once a week or so. The backup server, with virtually no load, surely doesn’t have a deadlock problem. I often attribute those deadlock problems to a disk access problem, as some times the network disk is terribly slow. Sometimes, it’s the high load of the computer that causes the problem. Sometimes, I can trace the bug to Java itself.”

The Deadlock Situation

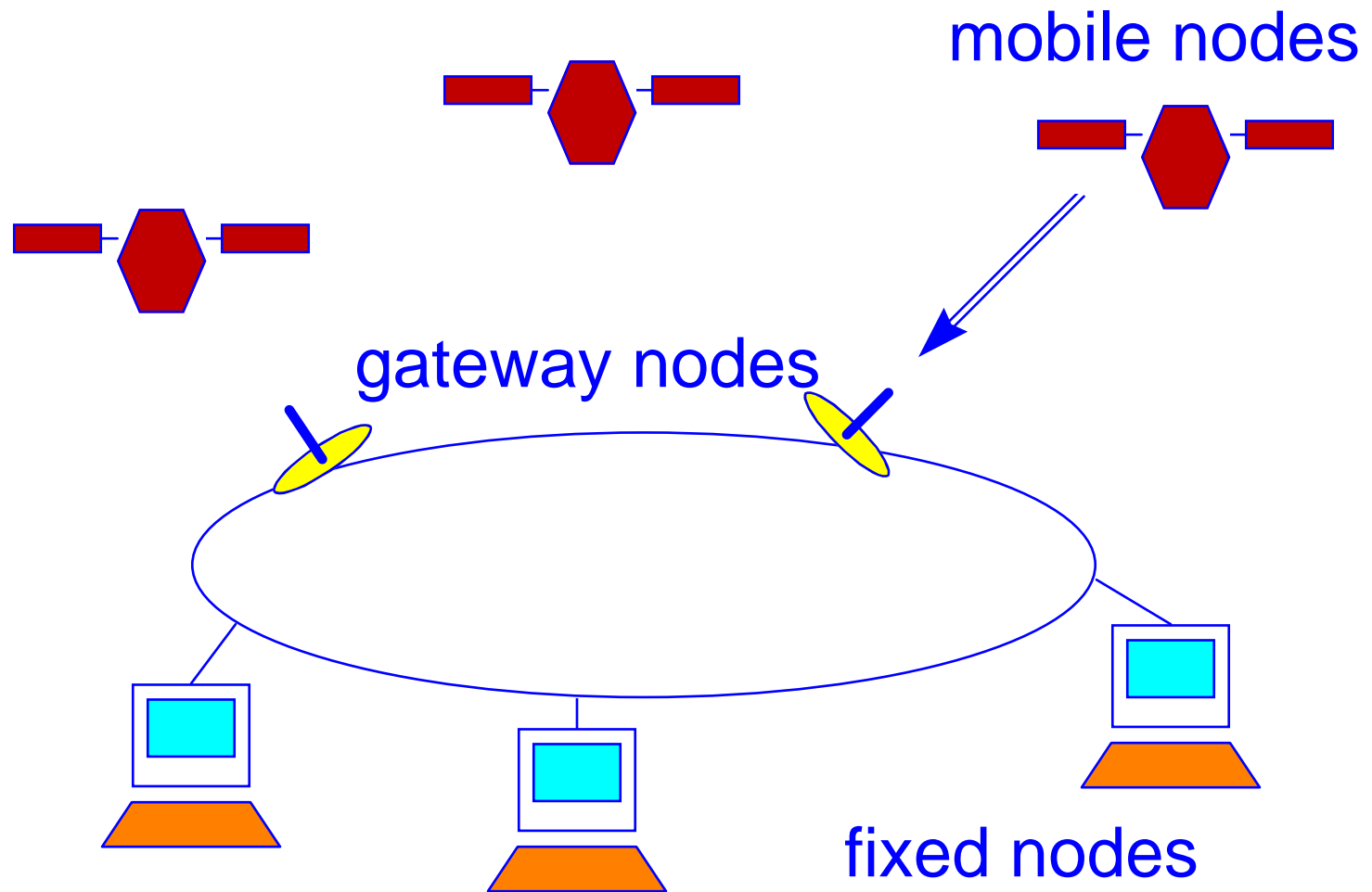


Simple Automatic File Exchange

Goddard Collaboration : SAFE

- *Supports low-cost spacecraft operation via the internet*
- *Provides a file transfer service between satellites and ground stations*
- *Works without operator to handle the unique space-ground communication challenges: mobility (satellites move) and intermittence (short contact frame)*

SAFE System Components



The Future

- *JPF1 : a prototype (proof of concept)*
- *JPF2 : next version within one year*
- *Supports unit verification directly*
- *Abstraction workbench for handling system verification (large programs)*
- *We are looking for applications to verify*